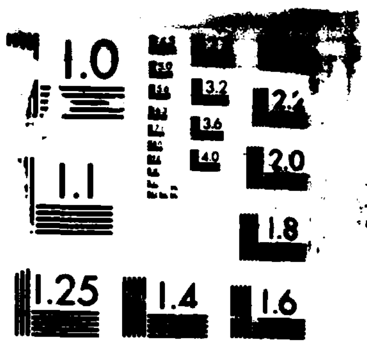
AD-A180 068 ADA (TRADENAME) COMPILER VALIDATION SUMMARY REPORT TOLERANT SYSTEMS TOLER..(U) INFORMATION SYSTEMS AND TECHNOLOGY CENTER W-P AFB OH ADA VALI.. 11 MAY 86 1/1

UNCLASSIFIED F/G 12/5 NL

END
DATE
FILMED
6 87


1.0
3.2
3.6
4.0
2.0
1.1
1.8
1.25
1.4
1.6

UNCLASSIFIED DTIC FILE COPY

(2)

SECURITY CLASSIFICATION OF THIS PAGE (*When Data Entered*)

AD-A180 068

| REPORT DOCUMENTATION PAGE | | READ INSTRUCTIONS BEFORE COMPLETEING FORM |
|---|---|---|
| 1. REPORT NUMBER | 2. GOVT ACCESSION NO. | 3. RECIPIENT'S CATALOG NUMBER |
| 4. TITLE (*and Subtitle*) Ada Compiler Validation Summary Report: Tolerant Systems, Tolerant Ada Development System (TX/VADS), Part Number S-240, Version 1.0, Tolerant ETernity | | 5. TYPE OF REPORT & PERIOD COVERED 11 MAY 1986 to 11 MAY 1987 |
| | | 6. PERFORMING ORG. REPORT NUMBER |
| 7. AUTHOR(s) Wright-Patterson | | 8. CONTRACT OR GRANT NUMBER(s) |
| 9. PERFORMING ORGANIZATION AND ADDRESS Ada Validation Facility ASD/SIOL Wright-Patterson AFB, OH 45433-6503 | | 10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS |
| 11. CONTROLLING OFFICE NAME AND ADDRESS Ada Joint Program Office United States Department of Defense Washington, DC 20301-3081 | | 12. REPORT DATE 11 MAY 1986 |
| | | 13. NUMBER OF PAGES |
| 14. MONITORING AGENCY NAME & ADDRESS(*If different from Controlling Office*) Wright-Patterson | | 15. SECURITY CLASS (*of this report*) UNCLASSIFIED |
| | | 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A |

16. DISTRIBUTION STATEMENT (*of this Report*)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (*of the abstract entered in Block 20. If different from Report*)

UNCLASSIFIED

DTIC ELECTE MAY 0 6 1987 S D E

18. SUPPLEMENTARY NOTES

19. KEYWORDS (*Continue on reverse side if necessary and identify by block number*)

Ada Programming language, Ada Compiler Validation Summary Report, Ada Compiler Validation Capability, ACVC, Validation Testing, Ada Validation Office, AVO, Ada Validation Facility, AVF, ANSI/MIL-STD-1815A, Ada Joint Program Office, AJPO

20. ABSTRACT (*Continue on reverse side if necessary and identify by block number*)

See Attached.

DD FORM 1473 EDITION OF 1 NOV 65 IS OBSOLETE
1 JAN 73 S/N 0102-LF-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (*When Data Entered*)

Ada® Compiler Validation Summary Report:

Compiler Name: Tolerant Ada Development System (TX™/VADS®),
Part Number S-240, Version 1.0

| Host Computer: | Target Computer: |
| --- | --- |
| Tolerant Eternity™ | Tolerant Eternity |
| under | under |
| TX 5.0.12 | TX 5.0.12 |

Testing Completed 11 May 1986 Using ACVC 1.7

This report has been reviewed and is approved.

Georgeanne Chitwood

Ada Validation Facility
Georgeanne Chitwood
ASD/SIOL
Wright-Patterson AFB, OH 45433-6503

Robert A. W... for JFK

Ada Validation Office
Dr. John F. Kramer
Institute for Defense Analyses
Alexandria VA

Virginia L. Castor

Ada Joint Program Office
Virginia L. Castor
Director
Department of Defense
Washington DC

| Accession For | |
| --- | --- |
| NTIS GRA&I | ☒ |
| DTIC TAB | ☐ |
| Unannounced | ☐ |
| Justification | |
| By | |
| Distribution/ | |
| Availability Codes | |
| Dist | Avail and/or Special |
| A-1 | |



®Ada is a registered trademark of the United States Government (Ada Joint Program Office).
™Eternity and TX are trademarks of Tolerant Systems.
®VADS is a registered trademark of the VERDIX Corporation.

AVF-VSR-30.0786

Ada® COMPILER
VALIDATION SUMMARY REPORT:
Tolerant Systems
Tolerant Ada Development System (TX™/VADS®),
Part Number S-240, Version 1.0
Tolerant Eternity™

Completion of On-Site Validation:
11 May 1986

Prepared By:
Ada Validation Facility
ASD/SIOL
Wright-Patterson AFB, OH 45433-6503

Prepared For:
Ada Joint Program Office
United States Department of Defense
Washington, D.C.

---

®Ada is a registered trademark of the United States Government (Ada Joint Program Office).
™Eternity and TX are trademarks of Tolerant Systems.
®VADS is a registered trademark of the VERDIX Corporation.

```
++++++++++++++++++++++++
+                      +
+ Place NTIS form here +
+                      +
++++++++++++++++++++++++
```

## EXECUTIVE SUMMARY

This Validation Summary Report (VSR) summarizes the results and conclusions of validation testing performed on the Tolerant Ada Development System (TX®/VADS®), Part Number S-240, Version 1.0 (hereafter refered to as the Tolerant Ada Development System), using Version 1.7 of the Ada® Compiler Validation Capability (ACVC).

The validation process includes submitting a suite of standardized tests (the ACVC) as inputs to an Ada compiler and evaluating the results. The purpose is to ensure conformance of the compiler to ANSI/MIL-STD-1815A Ada by testing that it properly implements legal language constructs and that it identifies and rejects illegal language constructs. The testing also identifies behavior that is implementation dependent but permitted by ANSI/MIL-STD-1815A. Six classes of tests are used. These tests are designed to perform checks at compile time, at link time, or during execution.

On-site testing was performed 5 May 1986 through 11 May 1986 at San Jose, CA. under the direction of the Ada Validation Facility, according to Ada Validation Organization (AVO) policies and procedures. The Tolerant Ada Development System is hosted on a Tolerant Eternity® operating under TX 5.0.12.

The results of validation are summarized in the following table:

| RESULT | TEST CLASS | | | | | | TOTAL |
|---|---|---|---|---|---|---|---|
| | A | B | C | D | E | L | |
| Passed | 68 | 819 | 1144 | 17 | 11 | 23 | 2082 |
| Failed | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Inapplicable | 0 | 5 | 176 | 0 | 0 | 0 | 181 |
| Withdrawn | 0 | 4 | 12 | 0 | 0 | 0 | 16 |
| TOTAL | 68 | 828 | 1332 | 17 | 11 | 23 | 2279 |

®Ada is a registered trademark of the United States Government (Ada Joint Program Office).
®Eternity and TX are trademarks of Tolerant Systems.
®VADS is a registered trademark of the VERDIX Corporation.

There were 16 withdrawn tests in ACVC Version 1.7 at the time of this validation attempt. A list of these tests appears in Appendix D.

Some tests demonstrate that some language features are or are not supported by an implementation. For this implementation, the tests determined the following:

- LONG_INTEGER and LONG_FLOAT are not supported.
- The additional predefined types TINY_INTEGER, SHORT_INTEGER, and SHORT_FLOAT are supported.
- Representation specifications for noncontiguous enumeration representations are supported.
- Generic unit specifications and bodies can be compiled in separate compilations.
- Pragma INLINE is supported for procedures and functions.
- The package SYSTEM is used by package TEXT_IO.
- Modes IN_FILE and OUT_FILE are supported for sequential I/O.
- Instantiation of the package SEQUENTIAL_IO with unconstrained array types is supported.
- Instantiation of the package SEQUENTIAL_IO with unconstrained record types with discriminants is supported.
- RESET and DELETE are supported for sequential and direct I/O.
- Modes IN_FILE, INOUT_FILE, and OUT_FILE are supported for direct I/O.
- Instantiation of package DIRECT_IO with unconstrained array types and unconstrained types with discriminants is supported.
- Dynamic creation and deletion of files are supported.
- More than one internal file can be associated with the same external file.
- An external file associated with more than one internal file can be reset.
- Illegal file names can exist.

ACVC Version 1.7 was taken on-site via magnetic tape to San Jose, CA. All tests, except the withdrawn tests and any executable tests that make use of a floating-point precision greater than SYSTEM.MAX_DIGITS, were compiled on

a Tolerant Eternity. Class A, C, D, and E tests were executed on a Tolerant Eternity.

On completion of testing, execution results for Class A, C, D, or E tests were examined. Compilation results for Class B were analyzed for correct diagnosis of syntax and semantic errors. Compilation and link results of Class L tests were analyzed for correct detection of errors.

The AVF identified 2093 of the 2279 tests in Version 1.7 of the ACVC as potentially applicable to the validation of Tolerant Ada Development System. Excluded were 170 tests requiring a floating-point precision greater than that supported by the implementation and the 16 withdrawn tests. After the 2093 tests were processed, 11 tests were determined to be inapplicable. The remaining 2082 tests were passed by the compiler.

The AVF concludes that these results demonstrate acceptable conformance to ANSI/MIL-STD-1815A.

TABLE OF CONTENTS


CHAPTER 1 INTRODUCTION

1.1 PURPOSE OF THIS VALIDATION SUMMARY REPORT . . . . 1-1
1.2 USE OF THIS VALIDATION SUMMARY REPORT . . . . . . . 1-2
1.3 RELATED DOCUMENTS . . . . . . . . . . . . . . . . . 1-3
1.4 DEFINITION OF TERMS . . . . . . . . . . . . . . . . 1-3
1.5 ACVC TEST CLASSES . . . . . . . . . . . . . . . . . 1-4

CHAPTER 2 CONFIGURATION INFORMATION

2.1 CONFIGURATION TESTED . . . . . . . . . . . . . . . . 2-1
2.2 CERTIFICATE INFORMATION . . . . . . . . . . . . . . 2-2
2.3 IMPLEMENTATION CHARACTERISTICS . . . . . . . . . . . 2-3

CHAPTER 3 TEST INFORMATION

3.1 TEST RESULTS . . . . . . . . . . . . . . . . . . . . 3-1
3.2 SUMMARY OF TEST RESULTS BY CLASS . . . . . . . . . . 3-1
3.3 SUMMARY OF TEST RESULTS BY CHAPTER . . . . . . . . . 3-2
3.4 WITHDRAWN TESTS . . . . . . . . . . . . . . . . . . 3-2
3.5 INAPPLICABLE TESTS . . . . . . . . . . . . . . . . . 3-2
3.6 SPLIT TESTS . . . . . . . . . . . . . . . . . . . . 3-3
3.7 ADDITIONAL TESTING INFORMATION . . . . . . . . . . . 3-4
3.7.1 Prevalidation . . . . . . . . . . . . . . . . . . 3-4
3.7.2 Test Method . . . . . . . . . . . . . . . . . . . 3-4
3.7.3 Test Site . . . . . . . . . . . . . . . . . . . . 3-5

APPENDIX A COMPLIANCE STATEMENT

APPENDIX B APPENDIX F OF THE Ada STANDARD

APPENDIX C TEST PARAMETERS

APPENDIX D WITHDRAWN TESTS

# CHAPTER 1

# INTRODUCTION

This Validation Summary Report (VSR) describes the extent to which a specific Ada compiler conforms to ANSI/MIL-STD-1815A. This report explains all technical terms used within it and thoroughly reports the results of testing this compiler using the Ada Compiler Validation Capability (ACVC). An Ada compiler must be implemented according to the Ada Standard (ANSI/MIL-STD-1815A). Any implementation-dependent features must conform to the requirements of the Ada Standard. The entire Ada Standard must be implemented, and nothing can be implemented that is not in the Standard.

Even though all validated Ada compilers conform to ANSI/MIL-STD-1815A, it must be understood that some differences do exist between implementations. The Ada Standard permits some implementation dependencies--for example, the maximum length of identifiers or the maximum values of integer types. Other differences between compilers result from limitations imposed on a compiler by the operating system and by the hardware. All of the dependencies demonstrated during the process of testing this compiler are given in this report.

VSRs are written according to a standardized format. The reports for several different compilers may, therefore, be easily compared. The information in this report is derived from the test results produced during validation testing. Additional testing information as well as details which are unique for this compiler are given in section 3.7. The format of a validation report limits variance between reports, enhances readability of the report, and minimizes the delay between the completion of validation testing and the publication of the report.

## 1.1 PURPOSE OF THIS VALIDATION SUMMARY REPORT

The VSR documents the results of the validation testing performed on an Ada compiler. Testing was carried out for the following purposes:

- To attempt to identify any language constructs supported by the compiler that do not conform to the Ada Standard

- To attempt to identify any unsupported language constructs required by the Ada Standard

- To determine that the implementation-dependent behavior is allowed by the Ada Standard

Testing of this compiler was conducted by SofTech, Inc. under the direction of the AVF according to policies and procedures established by the Ada Validation Organization (AVO). Testing was conducted from 5 May 1986 through 11 May 1986 at Tolerant Systems, Inc. in San Jose, CA.

## 1.2 USE OF THIS VALIDATION SUMMARY REPORT

Consistent with the national laws of the originating country, the AVO may make full and free public disclosure of this report. In the United States, this is provided in accordance with the "Freedom of Information Act" (5 U.S.C. #552). The results of this validation apply only to the computers, operating systems, and compiler versions identified in this report.

The organizations represented on the signature page of this report do not represent or warrant that all statements set forth in this report are accurate and complete, or that the subject compiler has no nonconformances to ANSI/MIL-STD-1815A other than those presented. Copies of this report are available to the public from:

Ada Information Clearinghouse
Ada Joint Program Office
OUSDRE
The Pentagon, Rm 3D-139
1211 S. Fern, C-107
Washington DC 20301-3081

or from:

Ada Validation Facility
ASD/SIOL
Wright-Patterson AFB, OH 45433-6503

Questions regarding this report or the validation test results should be directed to the AVF listed above or to:

Ada Validation Organization
Institute for Defense Analyses
1801 North Beauregard
Alexandria VA 22311

## 1.3 RELATED DOCUMENTS

1. Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A, FEB 1983.

2. Ada Validation Organization: Policies and Procedures, MITRE Corporation, JUN 1982, PB 83-110601.

3. Ada Compiler Validation Capability Implementers' Guide, SofTech, Inc., DEC 1984.

## 1.4 DEFINITION OF TERMS

| | |
|---|---|
| ACVC | The Ada Compiler Validation Capability. A set of programs that evaluates the conformance of a compiler to the Ada language specification, ANSI/MIL-STD-1815A. |
| Ada Standard | ANSI/MIL-STD-1815A, February 1983. |
| Applicant | The agency requesting validation. |
| AVF | The Ada Validation Facility. In the context of this report, the AVF is responsible for conducting compiler validations according to established policies and procedures. |
| AVO | The Ada Validation Organization. In the context of this report, the AVO is responsible for setting policies and procedures for compiler validations. |
| Compiler | A processor for the Ada language. In the context of this report, a compiler is any language processor, including cross-compilers, translators, and interpreters. |
| Failed test | A test for which the compiler generates a result that demonstrates nonconformance to the Ada Standard. |
| Host | The computer on which the compiler resides. |

| | |
|---|---|
| Inapplicable test | A test that uses features of the language that a compiler is not required to support or may legitimately support in a way other than the one expected by the test. |
| LMC | The Language Maintenance Committee whose function is to resolve issues concerning the Ada language. |
| Passed test | A test for which a compiler generates the expected result. |
| Target | The computer for which a compiler generates code. |
| Test | A program that evaluates the conformance of a compiler to a language specification. In the context of this report, the term is used to designate a single ACVC test. The text of a program may be the text of one or more compilations. |
| Withdrawn test | A test found to be inaccurate in checking conformance to the Ada language specification. A withdrawn test has an invalid test objective, fails to meet its test objective, or contains illegal or erroneous use of the language. |

## 1.5 ACVC TEST CLASSES

Conformance to *ANSI/MIL-STD-1815A* is measured using the Ada Compiler Validation Capability (ACVC). The ACVC contains both legal and illegal Ada programs structured into six test classes: A, B, C, D, E, and L. The first letter of a test name identifies the class to which it belongs. Special program units are used to report the results of the Class A, C, D, and E tests during execution. Class B tests are expected to produce compilation errors, and Class L tests are expected to produce link errors.

Class A tests check that legal Ada programs can be successfully compiled and executed. (However, no checks are performed during execution to see if the test objective has been met.) For example, a Class A test checks that reserved words of another language (other than those already reserved in the Ada language) are not treated as reserved words by an Ada compiler. A Class A test is passed if no errors are detected at compile time and the program executes to produce a message indicating that it has passed.

Class B tests check that a compiler detects illegal language usage. Class B tests are not executable. Each test in this class is compiled and the resulting compilation listing is examined to verify that every syntactical or semantic error in the test is detected. A Class B test is passed if every illegal construct that it contains is detected by the compiler.

Class C tests check that legal Ada programs can be correctly compiled and executed. Each Class C test is self-checking and produces a PASSED, FAILED, or NOT-APPLICABLE message indicating the result when it is executed.

Class D tests check the compilation and execution capacities of a compiler. Since there are no requirements placed on a compiler by the Ada Standard for some parameters (e.g., the number of identifiers permitted in a compilation, the number of units in a library, and the number of nested loops in a subprogram body), a compiler may refuse to compile a Class D test and still be a conforming compiler. Therefore, if a Class D test fails to compile because the capacity of the compiler is exceeded, the test is classified as inapplicable. If a Class D test compiles successfully, it is self-checking and produces a PASSED or FAILED message during execution.

Each Class E test is self-checking and produces a NOT-APPLICABLE, PASSED, or FAILED message when it is compiled and executed. However, the Ada Standard permits an implementation to reject programs containing some features addressed by Class E tests during compilation. Therefore, a Class E test is passed by a compiler if it is compiled successfully and executes to produce a PASSED message, or if it is rejected by the compiler for an allowable reason.

Class L tests check that incomplete or illegal Ada programs involving multiple, separately compiled units are detected and not allowed to execute. Class L tests are compiled separately and execution is attempted. A Class L test passes if it is rejected at link time--that is, an attempt to execute the main program must generate an error message before any declarations in the main program or any units referenced by the main program are elaborated.

Two library units, the package REPORT and the procedure CHECK_FILE, support the self-checking features of the executable tests. The package REPORT provides the mechanism by which executable tests report results. It also provides a set of identity functions used to defeat some compiler optimization strategies and force computations to be made by the target computer instead of by the compiler on the host computer. The procedure CHECK_FILE is used to check the contents of text files written by some of the Class C tests for chapter 14 of the Ada Standard.

The operation of these units is checked by a set of executable tests. These tests produce messages that are examined to verify that the units are operating correctly. If these units are not operating correctly, then the validation is not attempted.

Some of the conventions followed in the ACVC are intended to ensure that the tests are reasonably portable without modification. For example, the tests make use of only the basic set of 55 characters, contain lines with a maximum length of 72 characters, use small numeric values, and place features that may not be supported by all implementations in separate tests. However, some tests contain values that require the test to be customized according to implementation-specific values. The values used for this validation are listed in Appendix C.

A compiler must correctly process each of the tests in the suite and demonstrate conformance to the Ada Standard by either meeting the pass criteria given for the test or by showing that the test is inapplicable to the implementation. Any test that was determined to contain an illegal

language construct or an erroneous language construct is withdrawn from the ACVC and, therefore, is not used in testing a compiler. The nonconformant tests are given in Appendix D.

## CHAPTER 2

## CONFIGURATION INFORMATION

### 2.1 CONFIGURATION TESTED

The candidate compilation system for this validation was tested under the following configuration:

Compiler: Tolerant Ada Development System (TX/VADS),
Part Number S-240, Version 1.0

Test Suite: Ada Compiler Validation Capability, Version 1.7

Host Computer:

| | |
|---|---|
| Machine(s): | Tolerant Eternity |
| Operating System: | TX 5.0.12 |
| Memory Size: | 16,777,216 bytes |

Target Computer:

| | |
|---|---|
| Machine(s): | Tolerant Eternity |
| Operating System: | TX 5.0.12 |
| Memory Size: | 16,777,216 bytes |

## 2.2 CERTIFICATE INFORMATION

Base Configuration:

Compiler: Tolerant Ada Development System (TX/VADS),
Part Number S-240, Version 1.0

Test Suite: Ada Compiler Validation Capability, Version 1.7

Certificate Date: 16 June 1986

Host Computer:

Machine(s): Tolerant Eternity

Operating System: TX 5.0.12

Target Computer:

Machine(s): Tolerant Eternity

Operating System: TX 5.0.12

## 2.3 IMPLEMENTATION CHARACTERISTICS

One of the purposes of validating compilers is to determine the behavior of a compiler in those areas of the Ada Standard that permit implementations to differ. Class D and E tests specifically check for such implementation differences. However, tests in other classes also characterize an implementation. This compiler is characterized by the following interpretations of the Ada Standard:

- Nongraphic characters.

  Nongraphic characters are defined in the ASCII character set but are permitted in Ada programs, even within character strings. The compiler correctly recognizes these characters as illegal in Ada compilations. The characters are not printed in the output listing. (See test B26005A.)

- Capacities.

  The compiler correctly processes compilations containing loop statements nested to 65 levels, block statements nested to 65 levels, recursive procedures nested to 17 levels. It correctly processes a compilation containing 723 variables in the same declarative part. (See tests D55A03A through D55A03H, D56001B, D64005E through D64005G, and D29002K.)

- Universal integer calculations.

  An implementation is allowed to reject universal integer calculations having values that exceed SYSTEM.MAX_INT. This implementation does not reject such calculations and processes them correctly. (See tests D4A002A, D4A002B, D4A004A, and D4A004B.)

- Predefined types.

  This implementation supports the additional predefined types TINY_INTEGER, SHORT_INTEGER, and SHORT_FLOAT in the package STANDARD. (See tests B86001CR and B86001CP.)

- Based literals.

  An implementation is allowed to reject a based literal with a value exceeding SYSTEM.MAX_INT during compilation, or it may raise NUMERIC_ERROR during execution. This implementation rejects the test during compilation. (See test E24101A.)

- Array types.

  When an array type is declared with an index range exceeding the INTEGER'LAST values and with a component that is a null BOOLEAN array, this compiler raises NUMERIC_ERROR when the type is declared. (See tests E36202A and E36202B.)

A packed BOOLEAN array having a 'LENGTH exceeding INTEGER'LAST raises NUMERIC_ERROR when the array type is declared . (See test C52103X.)

A packed two-dimensional BOOLEAN array with more than INTEGER'LAST components raises NUMERIC_ERROR when the array type is declared . (See test C52104Y.)

A null array with one dimension of length greater than INTEGER'LAST may raise NUMERIC_ERROR either when declared or assigned. Alternately, an implementation may accept the declaration. However, lengths must match in array slice assignments. This implementation raises NUMERIC_ERROR when the array type is declared. (See test E52103Y.)

In assigning one-dimensional array types, the entire expression appears to be evaluated before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype.

In assigning two-dimensional array types, the entire expression does not appear to be evaluated before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

- Discriminated types.

During compilation, an implementation is allowed to either accept or reject an incomplete type with discriminants that is used in an access type definition with a compatible discriminant constraint. This implementation accepts such subtype indications during compilation. (See test E38104A.)

In assigning record types with discriminants, the entire expression appears to be evaluated before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

- Aggregates.

In the evaluation of a multi-dimensional aggregate, all choices appear to be evaluated before checking against the index type. (See tests C43207A and C43207B.)

In the evaluation of an aggregate containing subaggregates, all choices are evaluated before being checked for identical bounds. (See test E43212B.)

All choices are evaluated before CONSTRAINT_ERROR is raised if a bound in a nonnull range of a nonnull aggregate does not belong to an index subtype. (See test E43211B.)

- Functions.

  The declaration of a parameterless function with the same profile as an enumeration literal in the same immediate scope is rejected by the implementation. (See test E66001D.)

- Representation clauses.

  Enumeration representation clauses are not supported. (See test BC1002A.)

- Pragmas.

  The pragma INLINE is supported for procedures and functions. (See tests CA3004E and CA3004F.)

- Input/output.

  The package SEQUENTIAL_IO can be instantiated with unconstrained array types and record types with discriminants. The package DIRECT_IO can be instantiated with unconstrained array types and record types with discriminants without defaults. (See tests CE2201D, CE2201E, and CE2401D.)

  More than one internal file can be associated with each external file for sequential I/O for both reading and writing. (See tests CE2107A .. CE2107F (6 tests).)

  More than one internal file can be associated with each external file for direct I/O for both reading and writing. (See tests CE2107A .. CE2107F (6 tests).)

  An external file associated with more than one internal file can be deleted. (See test CE2110B.)

  More than one internal file can be associated with each external file for text I/O for both reading and writing. (See tests CE3111A .. CE3111E (5 tests).)

  An existing text file can be opened in OUT_FILE mode, can be created in OUT_FILE mode, and can be created in IN_FILE mode. (See test EE3102C.)

  Temporary sequential files are given a name. Temporary direct files are given a name. Temporary files given names are deleted when they are closed. (See test CE2108A.)

CHAPTER 3

# TEST INFORMATION

## 3.1 TEST RESULTS

The AVF identified 2093 of the 2279 tests in Version 1.7 of the Ada Compiler Validation Capability as potentially applicable to the validation of Tolerant Ada Development System, Part Number S-240, Version 1.0. Excluded were 170 tests requiring a floating-point precision greater than that supported by the implementation and the 16 withdrawn tests. After they were processed, 11 tests were determined to be inapplicable. The remaining 2082 tests were passed by the compiler.

The AVF concludes that the testing results demonstrate acceptable conformance to the Ada Standard.

## 3.2 SUMMARY OF TEST RESULTS BY CLASS

| RESULT | TEST CLASS | | | | | | TOTAL |
|---|---|---|---|---|---|---|---|
| | A | B | C | D | E | L | |
| Passed | 68 | 819 | 1144 | 17 | 11 | 23 | 2082 |
| Failed | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Inapplicable | 0 | 5 | 176 | 0 | 0 | 0 | 181 |
| Withdrawn | 0 | 4 | 12 | 0 | 0 | 0 | 16 |
| TOTAL | 68 | 828 | 1332 | 17 | 11 | 23 | 2279 |

## 3.3 SUMMARY OF TEST RESULTS BY CHAPTER

| RESULT | CHAPTER 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 14 | TOTAL |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Passed | 102 | 234 | 308 | 244 | 161 | 97 | 157 | 198 | 105 | 28 | 216 | 232 | 2082 |
| Failed | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Inapplicable | 14 | 73 | 86 | 3 | 0 | 0 | 4 | 1 | 0 | 0 | 0 | 0 | 181 |
| Withdrawn | 0 | 1 | 4 | 0 | 0 | 0 | 1 | 2 | 6 | 0 | 1 | 1 | 16 |
| TOTAL | 116 | 308 | 398 | 247 | 161 | 97 | 162 | 201 | 111 | 28 | 217 | 233 | 2279 |

## 3.4 WITHDRAWN TESTS

The following tests have been withdrawn from the ACVC Version 1.7:

| | | |
|---|---|---|
| B4A010C | C41404A | CA1003B |
| B83A06B | C48008A | CA3005A through CA3005D (4 tests) |
| BA2001E | C4A014A | CE2107E |
| BC3204C | C92005A | |
| C35904A | C940ACA | |

See Appendix D for the test descriptions.

## 3.5 INAPPLICABLE TESTS

Some tests do not apply to all compilers because they make use of features that a compiler is not required by the Ada Standard to support. Others may depend on the result of another test that is either inapplicable or withdrawn. For this validation attempt, 181 tests were inapplicable for the reasons indicated:

- C34001E, B52004D, B55B09C, B86001CS, and C55B07A use LONG_INTEGER which is not supported by this compiler.
- C34001G, C35702B, and B86001CQ use LONG_FLOAT which is not supported by this compiler.
- B86001DT requires a predefined numeric type other than those defined by the Ada language in package STANDARD. There is no such type for this implementation.
- C86001F redefines package SYSTEM, but TEXT_IO is made obsolete by this new definition in this implementation.

- C96005B checks implementations for which the smallest and largest values in type DURATION are different from the smallest and largest values in DURATIONS's base type. This is not the case for this implementation.

- 170 tests were not processed because SYSTEM.MAX_DIGITS was 15. These tests were:

C24113L through C24113Y (14 tests)
C35705L through C35705Y (14 tests)
C35706L through C35706Y (14 tests)
C35707L through C35707Y (14 tests)
C35708L through C35708Y (14 tests)
C35802L through C35802Y (14 tests)
C45241L through C45241Y (14 tests)
C45321L through C45321Y (14 tests)
C45421L through C45421Y (14 tests)
C45424L through C45424Y (14 tests)
C45521L through C45521Z (15 tests)
C45621L through C45621Z (15 tests)

## 3.6 SPLIT TESTS

If one or more errors do not appear to have been detected in a Class B test because of compiler error recovery, then the test is split into a set of smaller tests that contain the undetected errors. These splits are then compiled and examined. The splitting process continues until all errors are detected by the compiler or until there is exactly one error per split. Any Class A, Class C, or Class E test that cannot be compiled and executed because of its size is split into a set of smaller subtests that can be processed.

Splits were required for 18 Class B tests.

| | | |
|---|---|---|
| B24104A | B33004A | B67001B |
| B24104B | B37201A | B67001C |
| B24104C | B38008A | B67001D |
| B2A003A | B44001A | B910ABA |
| B2A003B | B64001A | B95001A |
| B2A003C | B67001A | B97101E |

## 3.7 ADDITIONAL TESTING INFORMATION

### 3.7.1 Prevalidation

Prior to validation, a set of test results for ACVC Version 1.7 produced by Tolerant Ada Development System was submitted to the AVF by the applicant for prevalidation review. Analysis of these results demonstrated that the compiler successfully passed all applicable tests.

### 3.7.2 Test Method

Testing of Tolerant Ada Development System using ACVC Version 1.7 was conducted on-site by a validation team. The configuration consisted of a Tolerant Eternity host and target operating under TX 5.0.12.

A magnetic tape containing ACVC Version 1.7 was taken on-site by the validation team. The magnetic tape contained all tests applicable to this validation, as well as all tests inapplicable to this validation except for any Class C tests that require floating-point precision exceeding the maximum value supported by the implementation. Tests that make use of values that are specific to an implementation were customized before being written to the magnetic tape. Tests requiring splits during the prevalidation testing were included in their split form on the magnetic tape. No editing of the test files was necessary when the validation team arrived on-site.

The contents of the magnetic tape were loaded directly onto the Tolerant Eternity. After the test files were loaded to disk, the full set of tests was compiled on the Tolerant Eternity, and all executable tests were run on the Tolerant Eternity. Results were transferred via ethernet from the Tolerant Eternity to a VAX-11/780 to be printed. Tests that were withdrawn from ACVC Version 1.7 were not run.

The compiler was tested using command scripts provided by Tolerant Systems. These scripts were reviewed by the validation team. Those which compile and link used the following options:

| | |
|---|---|
| -M <unit_name> | Produce an executable unit using <unit_name> as the main program. |
| -o <a.out> | Output_filename is <a.out>. |

Tests were run in batch mode using a single host and target computer. Test output, compilation listings, and job logs were captured on magnetic tape and archived at the AVF. The listings examined on-site by the validation team were also archived.

### 3.7.3 Test Site

The validation team arrived at San Jose, CA on 5 May 1986 and departed after testing was completed on 11 May 1986.

## APPENDIX A

## COMPLIANCE STATEMENT

Tolerant Systems has submitted the following compliance statement concerning the Tolerant Ada Development System.

## Compliance Statement

Base Configuration:

Compiler: Tolerant Ada Development System, TX/VADS, part number S-240, Version 1.0

Test Suite: Ada Compiler Validation Capability, Version 1.7

Host Computer:

| | |
|---|---|
| Machine: | Tolerant Eternity |
| Operating System: | TX 5.0.12 |

Tolerant Systems has made no deliberate extensions to the Ada language standard.

Tolerant Systems agrees to the public disclosure of this report.

Tolerant Systems agrees to comply with the Ada trademark policy, as defined by the Ada Joint Program Office.

Dale L. Shipley    Dated May 5, 1986

Tolerant Systems
Dale Shipley
Vice President, Engineering

APPENDIX B

APPENDIX F OF THE Ada STANDARD

1. Implementation-dependent Pragmas

Tolerant ADS provides for sharing of generic bodies (procedures and packages), when the generic parameters are restricted to enumeration types, integer types, and floating types.

PRAGMA SHARE_BODY is used to indicate desire to share or not share an instantiation. The pragma may reference the generic unit or the instantiated unit. When it references a generic unit, it sets sharing on/off for all instantiations of that generic, unless overridden by specific SHARE_BODY pragmas for individual instantiations. When it references an instantiated unit, sharing is on/off only for that unit. The default is to share all generics that can be shared, unless the unit uses PRAGMA IN_LINE.

PRAGMA SHARE_BODY is only allowed in the following places: immediately within a declarative part, immediately within a package specification, or after a library unit in a compilation, but before any subsequent compilation unit. The form of this pragma is

```
pragma SHARE_BODY (generic_name, boolean_literal)
```

Note that a parent instantiation is independent of any individual instantiation, therefore recompilation of a generic with different parameters has no effect on other compilations that reference it. The unit that caused compilation of a parent instantiation need not be referenced in any way by subsequent units that share the parent instantiation.

Sharing generics causes a slight execution time penalty because all type attributes must be indirectly referenced (as if an extra calling argument were added). However, it substantially reduces compilation time in most circumstances and reduces program size.

Tolerant has compiled a unit, SHARED_IO, in the standard library that instantiates all Ada generic I/O packages. Thus, any instantiation of an Ada I/O generic package will share one of the parent instantiation generic bodies. The PRAGMA SHARE_BODY takes the name of a generic instantiation or a generic unit as the first argument and one of the identifiers TRUE or FALSE as the second argument. This pragma is only allowed immediately at the place of a declarative item in a declarative part or package specification, or after a library unit in a compilation, but before any subsequent compilation unit.

When the first argument is a generic unit, the pragma applies to all instantiations of that generic. When the first argument is the name of a generic instantiation the pragma applies only to the specified instantiation, or overloaded instantiation.

If the second argument is TRUE, the compiler will try to share code generated for a generic instantiation with code generated for other instantiations of the same generic. When the second argument is FALSE, each instantiation will get a unique copy of the generated code. The extent to which code is shared between instantiations depends on this pragma and the kind of generic formal parameters declared for the generic unit.

PRAGMA EXTERNAL_NAME allows variables defined in another language to be referenced directly in Ada. PRAGMA EXTERNAL_NAME will replace all occurrences of variable_name with an external reference to link_name in the object file using the format shown below.

```
pragma EXTERNAL_NAME (variable_name, link_name);
```

This pragma is allowed at the place of a declarative item in a package specification and must apply to an object declared earlier in the same package specification. The object must be declared as a scalar or an access type. The object cannot be any of the following

a loop variable,
a constant,
an initialized variable,
an array, or
a record.

The link_name must be constructed as expected by the linker ld(1). For example, if linking with a C program on UNIX, the C variable name preceded by an underscore must be used in the same case (upper and lower) as in the C program source file, i.e., to link to the C global variable errno.

```
package PACKAGE_NAME is
 . . .
 ERRNO:INTEGER;
 pragma EXTERNAL_NAME(ERRNO,"_errno");
 . . .
end PACKAGE_NAME;
```

2. Implementation-dependent Attributes

There are no implementation-dependent attributes in Tolerant ADS.

4. Restrictions on Representation Clauses

4.1. PRAGMA PACK

Bit packing is not supported. Objects and components are packed to the nearest whole STORAGE_UNIT.

4.2. Size Specification

The size specification T'SMALL is not supported.

4.3. Record Representation Clauses

Component clauses must be aligned on STORAGE_UNIT boundaries.

4.4. Address Clauses

Address clauses are not supported.

4.5 Interrupts

Interrupts are not supported.

4.6 Change of Representation

Change of representation is not supported for record types.

4.7 Representation Attributes

The ADDRESS attribute is not supported for the following entities: static constants packages tasks labels entries.

## 4.8. Machine Code Insertions

Machine code insertions are not supported.

## 5. Conventions for Implementation-generated Names

There are no implementation generated names.

## 6. Interpretation of Expressions in Address Clauses

Address clauses are not supported.

## 7. Restrictions on Unchecked Conversions

The predefined generic function UNCHECKED_CONVERSION cannot be instantiated with a target type that is an unconstrained array type or an unconstrained record type with discriminants.

## 8. Implementation Characteristics of I/O Packages

Instantiations of DIRECT_IO use the value MAX_REC_SIZE as the record size (expressed in STORAGE_UNITs) when the size of ELEMENT_TYPE exceeds that value. For example, for unconstrained arrays such as string where ELEMENT_TYPE'SIZE is very large, MAX_REC_SIZE is used instead. MAX_RECORD_SIZE is defined in SYSTEM and can be changed by a program before instantiating DIRECT_IO to provide an upper limit on the record size. In any case, the maximum size supported is 1024 * 1024 * STORAGE_UNIT bits. DIRECT_IO will raise USE_ERROR if MAX_REC_SIZE exceeds this absolute limit.

Instantiations of SEQUENTIAL_IO use the value MAX_REC_SIZE as the record size (expressed in STORAGE_UNITs) when the size of ELEMENT_TYPE exceeds that value. For example, for unconstrained arrays such as string where ELEMENT_TYPE'SIZE is very large, MAX_REC_SIZE is used instead. MAX_RECORD_SIZE is defined in SYSTEM and can be changed by a program before instantiating INTEGER_IO to provide an upper limit on the record size. SEQUENTIAL_IO imposes no limit on MAX_REC_SIZE.

```
package standard is

   type boolean is (false, true);
   function "="        (left, right: boolean) return boolean;
   function "/="       (left, right: boolean) return boolean;
   function "<"        (left, right: boolean) return boolean;
   function "<="       (left, right: boolean) return boolean;
   function ">"        (left, right: boolean) return boolean;
   function ">="       (left, right: boolean) return boolean;
   function "and"      (left, right: boolean) return boolean;
   function "or"       (left, right: boolean) return boolean;
   function "xor"      (left, right: boolean) return boolean;
   function "not"      (right: boolean) return boolean;

   type tiny_integer is range -128 ..  127;
   function "="        (left, right: tiny_integer) return boolean;
   function "/="       (left, right: tiny_integer) return boolean;
   function "<"        (left, right: tiny_integer) return boolean;
   function "<="       (left, right: tiny_integer) return boolean;
   function ">"        (left, right: tiny_integer) return boolean;
   function ">="       (left, right: tiny_integer) return boolean;
   function "+"        (right: tiny_integer) return tiny_integer;
   function "-"        (right: tiny_integer) return tiny_integer;
   function "abs       (right: tiny_integer) return tiny_integer;
   function "+"        (left, right: tiny_integer) return tiny_integer;
   function "-"        (left, right: tiny_integer) return tiny_integer;
   function "*"        (left, right: tiny_integer) return tiny_integer;
   function "/"        (left, right: tiny_integer) return tiny_integer;
   function "rem"      (left, right: tiny_integer) return tiny_integer;
   function "mod"      (left, right: tiny_integer) return tiny_integer;
   function "**"       (left, right: tiny_integer) return tiny_integer;

   type short_integer is range -32768 .. 32767;
   function "="        (left, right: short_integer) return boolean;
   function "/="       (left, right: short_integer) return boolean;
   function "<"        (left, right: short_integer) return boolean;
   function "<="       (left, right: short_integer) return boolean;
   function ">"        (left, right: short_integer) return boolean;
   function ">="       (left, right: short_integer) return boolean;
   function "+"        (right: short_integer) return short_integer;
   function "-"        (right: short_integer) return short_integer;
   function "abs       (right: short_integer) return short_integer;
   function "+"        (left, right: short_integer) return short_integer;
   function "-"        (left, right: short_integer) return short_integer;
   function "*"        (left, right: short_integer) return short_integer;
   function "/"        (left, right: short_integer) return short_integer;
   function "rem"      (left, right: short_integer) return short_integer;
   function "mod"      (left, right: short_integer) return short_integer;
   function "**"       (left, right: short_integer) return short_integer;

   type integer is  range -2147483648 ..  2147483647;
   function "="        (left, right: integer) return boolean;
   function "/="       (left, right: integer) return boolean;
   function "<"        (left, right: integer) return boolean;
   function "<="       (left, right: integer) return boolean;
```

```
function ">"        (left, right: integer) return boolean;
function ">="       (left, right: integer) return boolean;
function "+"        (right: integer) return integer;
function "-"        (right: integer) return integer;
function "abs       (right: integer) return integer;
function "+"        (left, right: integer) return integer;
function "-"        (left, right: integer) return integer;
function "*"        (left, right: integer) return integer;
function "/"        (left, right: integer) return integer;
function "rem"      (left, right: integer) return integer;
function "mod"      (left, right: integer) return integer;
function "**"       (left, right: integer) return integer;

type short_float is digits 6 range
    -2#1.11111_11111_11111_11111_111#E127 ..
     2#1.11111_11111_11111_11111_111#E127;
function "="        (left, right: short_float) return boolean;
function "/="       (left, right: short_float) return boolean;
function "<"        (left, right: short_float) return boolean;
function "<="       (left, right: short_float) return boolean;
function ">"        (left, right: short_float) return boolean;
function ">="       (left, right: short_float) return boolean;
function "+"        (right: short_float) return short_float;
function "-"        (right: short_float) return short_float;
function "abs       (right: short_float) return short_float;
function "+"        (left, right: short_float) return short_float;
function "-"        (left, right: short_float) return short_float;
function "*"        (left, right: short_float) return short_float;
function "/"        (left, right: short_float) return short_float;
function "**"       (left, right: short_float) return short_float;

type float is digits 15 range
  -2#1.111111111_111111111_111111111_111111111_111111111_11#E1023 ..
   2#1.111111111_111111111_111111111_111111111_111111111_11#E1023;
function "="        (left, right: float) return boolean;
function "/="       (left, right: float) return boolean;
function "<"        (left, right: float) return boolean;
function "<="       (left, right: float) return boolean;
function ">"        (left, right: float) return boolean;
function ">="       (left, right: float) return boolean;
function "+"        (right: float) return float;
function "-"        (right: float) return float;
function "abs       (right: float) return float;
function "+"        (left, right: float) return float;
function "-"        (left, right: float) return float;
function "*"        (left, right: float) return float;
function "/"        (left, right: float) return float;
function "**"       (left, right: float) return float;
function "*"        (left: univ_integer; right: univ_real) return univ_re
function "*"        (left: univ_real; right: univ_integer) return univ_re
function "/"        (left: univ_real; right: univ_integer) return univ_re
function "*"        (left: any_fixed; right: any_fixed) return univ_fixed
function "/"        (left: any_fixed; right: any_fixed) return univ_fixed

type character is
    (nul, soh, stx, etx, eot, enq, eck, bel,
     bs,  ht,  lf,  vt,  ff,  cr,  so,  si,
     dle, dcl, dc2, dc3, dc4, nak, syn, etb,
```

```
        can, em,  sub, esc, fs,  gs,  rs,  us,
        ' ', '!', '"', '#', '$', '%', '&', ''',
        '(', ')', '*', '+', ',', '-', '.', '/',
        '0', '1', '2', '3', '4', '5', '6', '7',
        '8', '9', ':', ';', '<', '=', '>', '?',
        '@', 'A', 'B', 'C', 'D', 'E', 'F', 'G',
        'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O',
        'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W',
        'X', 'Y', 'Z', '[', '\', ']', '^', '_',
        '`', 'a', 'b', 'c', 'd', 'e', 'f', 'g',
        'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o',
        'p', 'q', 'r', 's', 't', 'u', 'v', 'w',
        'x', 'y', 'z', '{', '|', '}', '~', del);

for character use
    (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, .... 127);

package ascii is
    nul: constant character := nul; soh: constant character := soh;
    stx: constant character := stx; etx: constant character := etx;
    eot: constant character := eot; enq: constant character := enq;
    ack: constant character := ack; bel: constant character := bel;
    lf : constant character := lf;  vt : constant character := vt;
    ff : constant character := ff;  cr : constant character := cr;
    so : constant character := so;  si : constant character := si;
    dle: constant character := dle; dc1: constant character := dc1;
    dc2: constant character := dc2; dc3: constant character := dc3;
    dc4: constant character := dc4; nak: constant character := nak;
    syn: constant character := syn; etb: constant character := etb;
    sub: constant character := sub; esc: constant character := esc;
    rs : constant character := rs;  us : constant character := us;
    del: constant character := del;
    exclam: constant character := '!';
    quotation: constant character := '"';
    sharp: constant character := '#';
    dollar: constant character := '$';
    percent: constant character := '%';
    ampersand: constant character := '&';
    colon: constant character := ':';
    semicolon: constant character := ';';
    query: constant character := '?';
    at_sign: constant character := '@';
    l_bracket: constant character := '[';
    back_slash: constant character := '\';
    r_bracket: constant character := ']';
    underline: constant character := '_';
    grave: constant character := '`';
    l_brace: constant character := '{';
    bar: constant character := '|';
    r_brace: constant character := '}';
    tilde: constant character := '~';

    lc_a: constant character := 'a';
    ...
    lc_z: constant character := 'z';

end ascii;
```

```
    subtype natural is integer range 0 .. integer'last;
    subtype positive is integer range 1 .. integer'last;

    type string is array(positive range <>) of character;
    pragma pack(string);
    function "="        (left, right: string) return boolean;
    function "/="       (left, right: string) return boolean;
    function "<"        (left, right: string) return boolean;
    function "<="       (left, right: string) return boolean;
    function ">"        (left, right: string) return boolean;
    function ">="       (left, right: string) return boolean;
    function "&"        (left: string; right: string) return string;
    function "&"        (left: character; right: string) return string;
    function "&"        (left: string; right: character) return string;
    function "&"        (left: character; right: character) return string;

    type duration is delta 2#1.0#E-14 range
        -2#1000000000000000000.0# ..
         2#11111111111111111.11111111111111#;
    function "="        (left, right: duration) return boolean;
    function "/="       (left, right: duration) return boolean;
    function "<"        (left, right: duration) return boolean;
    function "<="       (left, right: duration) return boolean;
    function ">"        (left, right: duration) return boolean;
    function ">="       (left, right: duration) return boolean;
    function "*"        (left: duration; right: integer) return duration;
    function "*"        (left: integer; right: duration) return duration;
    function "/"        (left: duration; right: integer) return duration;

    constraint_error: exception;
    numeric_error: exception;
    program_error: exception;
    storage_error: exception;
    tasking_error: exception;
end standard;
```

## APPENDIX C

## TEST PARAMETERS

Certain tests in the ACVC make use of implementation-dependent values, such as the maximum length of an input line and invalid file names. A test that makes use of such values is identified by the extension .TST in its file name. Actual values to be substituted are identified by names that begin with a dollar sign. A value is substituted for each of these names before the test is run. The values used for this validation are given below.

| Name and Meaning | Value |
|---|---|
| $BIG_ID1<br>Identifier of size MAX_IN_LEN with varying last character. | (1..498=>'A', 499=>'1') |
| $BIG_ID2<br>Identifier of size MAX_IN_LEN with varying last character. | (1..498=>'A', 499=>'2') |
| $BIG_ID3<br>Identifier of size MAX_IN_LEN with varying middle character. | (1..249=>'A', 250=>'3', 251..499=>'A') |
| $BIG_ID4<br>Identifier of size MAX_IN_LEN with varying middle character. | (1..249=>'A', 250=>'4', 251..499=>'A') |
| $BIG_INT_LIT<br>An integer literal of value 298 with enough leading zeroes so that it is MAX_IN_LEN characters long. | (1..496=>'0', 497..499=>"298") |

TEST PARAMETERS

| Name and Meaning | Value |
|---|---|
| $BIG_REAL_LIT<br>A real literal that can be either of floating- or fixed-point type, has value 690.0, and has enough leading zeroes to be MAX_IN_LEN characters long. | (1..493=>'0', 494..499=>"69.0E1") |
| $BLANKS<br>Blanks of length MAX_IN_LEN - 20 | 479 |
| $COUNT_LAST<br>Value of COUNT'LAST in TEXT_IO package. | 2147483647 |
| $EXTENDED_ASCII_CHARS<br>A string literal containing all the ASCII characters with printable graphics that are not in the basic 55 Ada character set. | "abcdefghijklmnopqrstuvwxyz!$%?@[\]^`{}~" |
| $FIELD_LAST<br>Value of FIELD'LAST in TEXT_IO package. | 2147483647 |
| $FILE_NAME_WITH_BAD_CHARS<br>An illegal external file name that either contains invalid characters or is too long. | "\/illegal\/file_name\/2{[$%2102C.DAT" |
| $FILE_NAME_WITH_WILD_CARD_CHAR<br>An external file name that either contains a wild card character or is too long. | "\/illegal\/file_name\/CE2102C*.DAT" |
| $GREATER_THAN_DURATION<br>A universal real value that lies between DURATION'BASE'LAST and DURATION'LAST or any value in the range of DURATION. | 100_000.0 |
| $GREATER_THAN_DURATION_BASE_LAST<br>The universal real value that is greater than DURATION'BASE'LAST. | 10_000_000.0 |
| $ILLEGAL_EXTERNAL_FILE_NAME1<br>Illegal external file name. | "\/no\/such\/directory\/" & "ILLEGAL_EXTERNAL_FILE_NAME1" |
| $ILLEGAL_EXTERNAL_FILE_NAME2<br>Illegal external file names. | "\/no\/such\/directory\/" & "ILLEGAL_EXTERNAL_FILE_NAME2" |

| Name and Meaning | Value |
|---|---|
| $INTEGER_FIRST<br>The universal integer literal expression whose value is INTEGER'FIRST. | -2147483648 |
| $INTEGER_LAST<br>The universal integer literal expression whose value is INTEGER'LAST. | 2147483647 |
| $LESS_THAN_DURATION<br>A universal real value that lies between DURATION'BASE'FIRST and DURATION'FIRST or any value in the range of DURATION. | -100_000.0 |
| $LESS_THAN_DURATION_BASE_FIRST<br>The universal real value that is less than DURATION'BASE'FIRST. | -10_000_000.0 |
| $MAX_DIGITS<br>Maximum digits supported for floating-point types. | 15 |
| $MAX_IN_LEN<br>Maximum input line length permitted by the implementation. | 499 |
| $NAME<br>A name of a predefined numeric type other than FLOAT, INTEGER, SHORT_FLOAT, SHORT_INTEGER, LONG_FLOAT, or LONG_INTEGER. | TINY_INTEGER |
| $NEG_BASED_INT<br>A based integer literal whose highest order nonzero bit falls in the sign bit position of the representation for SYSTEM.MAX_INT. | 16#FFFFFFFD# |
| $NON_ASCII_CHAR_TYPE<br>An enumerated type definition for a character type whose literals are the identifier NON_NULL and all non-ASCII characters with printable graphics. | (NON_NULL) |

## APPENDIX D

## WITHDRAWN TESTS

Some tests are withdrawn from the ACVC because they do not conform to the Ada Standard. When testing was performed, the following 16 tests had been withdrawn at the time of validation testing for the reasons indicated:

- B4A010C: The object_declaration in line 18 follows a subprogram body of the same declarative part.

- B83A06B: The Ada Standard 8.3(17) and AI-00330 permit the label LAB_ENUMERAL of line 80 to be considered a homograph of the enumeration literal in line 25.

- BA2001E: The Ada Standard 10.2(5) states: "Simple names of all subunits that have the same ancestor library unit must be distinct identifiers." This test checks for the above condition when stubs are declared. However, the Ada Standard does not preclude the check being made when the subunit is compiled.

- BC3204C: The file BC3204C4 should contain the body for BC3204C0 as indicated in line 25 of BC3204C3M.

- C35904A: The elaboration of subtype declarations SFX3 and SFX4 may raise NUMERIC_ERROR (instead of CONSTRAINT_ERROR).

- C41404A: The values of 'LAST and 'LENGTH are incorrect in IF statements from line 74 to the end of the test.

- C48008A: This test requires that the evaluation of default initial values not occur when an exception is raised by an allocator. However, the Language Maintenance Committee (LMC) has ruled that such a requirement is incorrect (AI-00397/01).

WITHDRAWN TESTS

- C4A014A: The number declarations in lines 19-22 are incorrect because conversions are not static.

- C92005A: At line 40, "/=" for type PACK.BIG_INT is not visible without a USE clause for package PACK.

- C940ACA: This test assumes that allocated task TT1 will run prior to the main program, and thus assign SPYNUMB the value checked for by the main program; however, such an execution order is not required by the Ada Standard, so the test is erroneous.

- CA1003B: This test requires all of the legal compilation units of a file containing some illegal units to be compiled and executed. According to AI-00255, such a file may be rejected as a whole.

- CA3005A..D (4 tests): No valid elaboration order exists for these tests.

- CE2107E: This test has a variable, TEMP_HAS_NAME, that needs to be given an initial value of TRUE.

ATE

LMED

-8